Panorama: Optimizing Internet-scale Users' Routes from End to End

Geng Li, Shuihai Hu, and Kun Tan, *Huawei Technologies*

https://www.usenix.org/conference/atc24/presentation/li-geng

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

Open access to the Proceedings of the 2024 USENIX Annual Technical Conference is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Panorama: Optimizing Internet-scale Users' Routes from End to End

Geng Li*, Shuihai Hu, and Kun Tan

Huawei Technologies

Abstract

Network performance is critical to the user experience of many real-time interactive applications, such as video conferencing and live streaming. Empirical studies [46] show that transport latency over 300ms would become unacceptable, leading to significant user satisfaction declining. Unfortunately, due to the best-effort nature of Internet, such strict performance requirement can hardly be fully met. Despite continuous efforts have been made to improve the performance of Internet (e.g., overlay routing optimization and content delivery network), we are still far from delivering satisfying network performance for these applications. The stringent network requirements, the world-wide cross-continental network transfers, and the large-scale Internet-wide users, together make it a complex challenge to deliver ideal user experience for emerging real-time applications.

In this paper, we present *Panorama*, a scalable system for delivering desired user experience to real-time applications over a globally distributed overlay network. Specifically, *Panorama* takes a centralized approach with end-to-end multi-objective traffic engineering optimization to meet tight performance requirement for real-time applications, and in the meanwhile achieving good scalability with a batch of optimizations including intelligent measurement-based user/session grouping and parallelizable path calculation. We evaluate *Panorama* based on 81 million selected real-world traces in deployment environment with clients across 66 countries. The extensive evaluation demonstrates that *Panorama* can support a routing service for millions of users, while providing latency lower than 200ms for 96.34% of the communication sessions, and improving SLA satisfaction by up to 88.0%.

1 Introduction

In recent years, due to the rapid growth of video conferencing, online education and virtual entertainment, there has been an increasingly strong requirement for *high performance* and *scalable* real-time interactive services. On the one hand, driven by the continuous improvement in interactivity and video quality, the emerging applications impose much more stringent requirement on both latency (e.g., cloud game requires a persistent low latency within 200ms for acceptable user experience [14]) and throughput (e.g., 8K immersive videos require a throughput of over 60Mbps [43]). On the other hand, the number of interactive applications and their users increases exponentially. For example, daily active users of video conferencing products have increased 2,900% since the COVID-19 pandemic [38]. This calls for a scalable solution to offer high-quality real-time services in a cost-effective way.

It is well-known that the public Internet was originally designed for connectivity as a best-effort infrastructure, and thus can hardly meet such growing requirements. In the past few decades, tremendous efforts have been made in designing and building overlay networks, for the purpose of overcoming the Internet's inherent inefficiency and other limitations [25, 41]. On top of the overlay networks, many overlay routing schemes are proposed to achieve better end-to-end performance than that provided by the native Internet [1, 4, 5, 9, 22, 40]. There are also some other efforts, such as traffic engineering (TE) solutions [20, 21, 28, 51] and content delivery network (CDN) systems [8, 11, 16, 33, 35].

Despite these efforts, with a more in-depth analysis, we found that all existing solutions are ill-suited for our scenario. The requirements of supporting low-latency interactivity, high data volumes and Internet-wide user scale have not been jointly considered before. Existing solutions either had unmatched goals (e.g., SWAN [20] and B4 [21] targeted higher cross-datacenter link utilization), or did not fully consider the dynamic nature of interactive application from end to end (e.g., the CDN system [11] is mainly for improving the delivery of cacheable content from servers to clients). Hence they fell short of achieving the best possible quality-of-experience (QoE) for emerging real-time interactive applications (more detailed discussions in Section 2.3 and Section 7).

In summary, there are mainly three challenges in provid-

*Corresponding author: Geng Li (ligeng23@huawei.com)

ing desired performance for real-time interactive applications. First, persistently meeting the requirements of high throughput and low latency needs careful routing arrangement and traffic scheduling. Second, the communication is often worldwide, and thus a global joint optimization over a set of cross-continental networks is unavoidable. Third, handling Internet-wide millions of daily active users itself presents a fundamental scalability difficulty.

To address the above challenges simultaneously, in this paper, we present *Panorama*, a scalable system for delivering desired QoE to clients of real-time interactive applications over globally distributed overlay network. At its core, *Panorama* takes a centralized approach that performs end-to-end multi-object QoE optimization by jointly coordinating the routes along the communication paths. *Panorama* systematically achieves better scalability by dynamic user and session grouping based on real-time measurements, and a heuristic, parallelizable, but nearly optimal K-shortest path algorithm to compute feasible path sets that satisfy application requirements. Besides routing, *Panorama* also dynamically engineers traffic across different paths to support bursty/high throughput for real-time interactive applications.

Panorama has been implemented and deployed in a large-scale overlay network testbed. We evaluate it extensively using 81 million real-world traces in deployment environment from users across 66 countries worldwide. Our experimental evaluation demonstrates the ability of *Panorama* to scale to millions of users, and shows a substantial amount of performance gain. In particular, *Panorama* can limit the latency of 96.34% of the communication sessions below 200ms, and 99.98% of the sessions below 300ms. Besides latency, *Panorama* achieves better loss and jitter performance than other solutions, resulting in significant service-level agreement (SLA) satisfaction improvement by up to 88.0%. We also compare *Panorama* with another widely-used commercial real-time overlay networking service–Agora SD-RTN [1] in a small-scale testbed with 21 users from 18 countries. The results show that *Panorama* achieves lower latency than Agora in more than 80% cases, by up to 250 ms, and 30.6% improvement on the median latency. Lastly, *Panorama* can reduce bandwidth cost by 44.3%, and balance load by 54.5% compared with other solutions.

2 Background and Motivation

This section gives some background of real-time application and overlay networking, and the motivation of *Panorama*.

2.1 Real-time interactive application

Some Internet-based applications, such as video conferencing and online gaming, are featured by their real-time interactivity, in the sense that the transmitted content is generated based on users' interaction with the applications in real-time. As

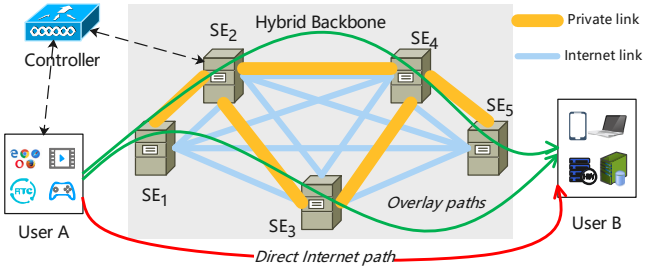


Figure 1: Overlay networking architecture.

content is *dynamic* and *uncacheable*, the user-perceived QoE is highly sensitive to the quality of network(s) that carries request/response between source(s) to and destination(s).

Driven by the continuous advances in technology, augmented reality (AR) and virtual reality (VR) systems are now being integrated with traditional real-time interactive applications for realizing immersive user experience. For instance, Meta is building its metaverse by augmenting its social media applications with immersive video support [31]. Given this trend, new challenges have emerged in terms of network latency and throughput. For example, to achieve acceptable immersive experience, a latency within 200ms and a throughput above 60Mbps is required [14]. Delivering ideal user experience is more challenging, i.e., 40ms latency and 100Mbps throughput is required. The scalability is also a big challenge. It is reported that, over 600 million people use Spark AR on Facebook and Instagram, and AR/VR will reach 25% of Internet users by 2023 [47].

2.2 Overlay networking

An overlay network is a logical network, built on top of the physical network, for providing richer network functionality and service. Overlay networking has been proposed for decades to overcome shortcomings of the IP routing infrastructures. It is first designed to improve fault recovery capability and reliability [4, 34, 39, 44], and then studied to provide better service performance [18, 19, 30, 42].

Figure 1 shows the architecture of a modern overlay network, which consists of a number of geographically distributed overlay nodes (called Service Edge (SE)). Each SE plays as an relay node that receives/forwards packets from/to users or other SEs with en/decapsulation. The connectivity among SEs can be hybrid, including private links and Internet links.

As depicted in the figure, SEs are usually controlled by a logically-centralized controller, and thus an overlay routing algorithm can be used to explicitly compute a set of overlay paths among communicating pairs. SE can do fine-grained active network measurement, making it possible to learn latency, bandwidth, loss and jitter of the hops along the paths connecting sources and destinations. This motivates our design to deliver desired network performance for real-time interactive

applications via end-to-end overlay routing optimization.

2.3 Limitation of existing solutions

Routing optimization over overlay network has been explored in a number of contexts [4, 5, 9]. Of interest to us are the ones to improve performance of service delivery. For the considerations of simplicity and scalability, existing commercially-deployed overlay networking systems widely adopt a "nearby access + backbone optimization" routing approach, which we refer to as *Nearby* solution for short. Such systems include Agora SD-RTN [1], Akamai overlay transport network [35, 41], AWS GA [15], Google Premium Network Service Tier [37], etc. The *Nearby* solution works typically in the following way: the source and destination users get associated with the closest SEs as ingress and egress respectively, and an optimized path between the ingress and egress is used to traverse the backbone.

The intuition behind *Nearby* solution is based on the assumption that the backbone network is typically private with significant more capacity and better performance than the public Internet. Therefore the *Nearby* solution can minimize the Internet usage and takes advantage of the backbone as much as possible. While being simple to scale, such independent routing optimizations across multiple networks is always non optimal from end to end, i.e., *Nearby* solution can fail to satisfy the strict end-to-end performance requirement (e.g., 200ms) for real-time interactive applications.

Via [22] is a recent research effort that takes a global view to jointly optimize end-to-end network performance for audio calls. It leverages client-side passive measurement, and combines prediction-based filtering with an online exploration-exploitation strategy to select desired relay paths for audio calls. Via is the closest work to us and partially inspires our design. The major limitation of Via is that, it was specifically designed for voice calls, and only provides the routing solution (without TE), as bandwidth and throughput were not considered due to the low data rate of audio call streams. Such design fails to support high-throughput applications (e.g., VR/AR) requirement.

3 System Design

In this section, we present an overview of *Panorama* system, including the architecture, workflow, and some components to handle scalability and dynamicity.

3.1 System architecture

Figure 2 depicts the system architecture, and key components.

***Panorama* Global Controller.** *Panorama* runs a logically centralized control plane to compute and manage the overlay routing for all communication sessions. The Global Controller

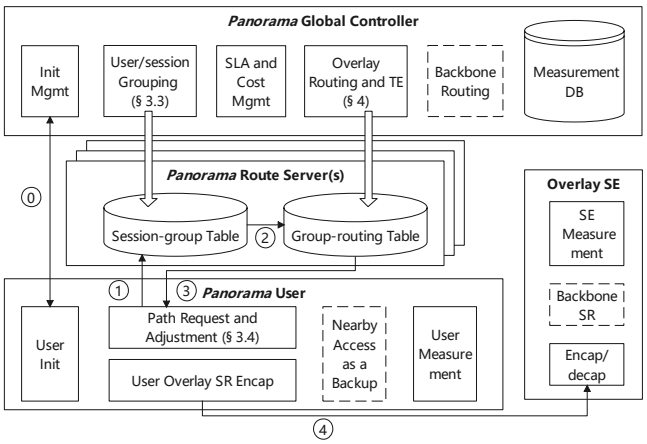


Figure 2: System architecture of *Panorama*.

itself is a fault-tolerant distributed platform that is able to survive failures of multiple data centers without interruption.

Init Mgmt manages the activities in user initialization process, including user authorization, application management, Route Server allocation, and so forth. *User/session Grouping* divides IP addresses and sessions of the whole network into equivalent classes in order to handle the scalability challenge. The grouping is not static, but changes adaptively based on measurement (details in Section 3.3). *SLA and Cost Mgmt* is used for operators to configure related parameters for overlay routing. *Overlay Routing and TE* (Section 4) is the main component in *Panorama* system, which computes the optimal path allocation in real time. *Backbone Routing* computes the optimal backbone path between any SE pair. This component belongs to the previous *Nearby* routing solution, and now is used as a backup in *Panorama*. *Measurement DB*, collecting measurement from both SEs and users, creates a real-time, topological Internet map capturing and estimating the state and quality of any connectivity across the overlay network. This map is an essential input for Routing and TE.

***Panorama* Route Server.** To offload routing queries from millions of users to the global controller, as well as to bring the route request service closer to users, a set of Route Servers are deployed globally. It provides two main functions: (1) cache of readily computed grouping and routing decisions from the Global Controller, and (2) efficient lookup service to quickly response path requests. Most of the Route Servers are co-located with SEs. This deployment design reduces operational cost by efficiently reusing the overlay infrastructures and control channels.

- *Session-group Table* maps any session to a session group (SG) that the session belongs to. This table is updated by the User/session Grouping component in the Global Controller, at a relatively low frequency (e.g., every tens of hours).
- *Group-routing Table* maps an SG ID to a weighted set paths for each SG, describing how incoming sessions in the SG should be mapped onto those paths. This table is updated by

the Routing and TE component, at a high frequency (e.g., every 5 minutes).

***Panorama* User.** Each user exchanges path information with Route Servers by HTTP protocols through the *Path Request and Adjustment* interface (Section 3.4). This component requests for an optimal path when a session is first created, and dynamically adjusts path during the session. *User Overlay SR Encap* is a data plane component that encodes an overlay path into the header of each packet, and sends the packets via overlay source routing (SR). *User Measurement* leverages Real User Measurement (RUM) techniques to measure user-side performance. The measurement results are used to evaluate our routing approaches, and provide monitoring data for application developers.

Overlay Service Edge (SE). Each SE runs a *SE Measurement* module that sends active probes to other SEs and endpoints on the Internet to collect information of latency, jitter, loss, and throughput, which is the main source of the Measurement DB. Another basic component in SE is the *Encap/decap* to support SR, relays packets based on the overlay path inside the packet header.

3.2 API and Workflow

The functions of *Panorama* are invoked by its APIs, among which the two routing-related APIs are `Pano_INIT` for initialization and `Pano_TX` for sending data. In every application program context, a *Panorama* instance maintains the user states and the callback functions, which is configured by `Pano_INIT` during the process of initialization. To send data by *Panorama* service, the application program first creates a local *sockfd*, and then calls `Pano_TX` with *sockfd* as parameters to create a RTC session. In this way, the following packets in sockfd will be associated in the session and sent via *Panorama*.

The steps in following workflows are labeled in Figure 2.

⓪ When `Pano_INIT` API is first invoked, the *Panorama* user interacts with the Init Mgmt component at Global Controller to download init configurations, including the user's nearby Route Server address and nearby SEs (for *Nearby* access), and get updated in every 30 minutes.

① When `Pano_TX` API is invoked, the user sends a route request to the Route Server, attached with the session information.

② Route Server looks up the two-table pipeline and determines a weighted path set.

③ A weighted random selection process, which is omitted in the figure, eventually selects one path and returns to the user.

④ The user then sends data via overlay SR according to path decision.

One may notice *Panorama* may incur an extra latency when requesting the path. In practice, such latency is negligible because (1) the request latency is 111.67ms on average, and each request has 200ms timeout to fall back to *Nearby* routing as a backup, and (2) it is a one-shot thing when creating a socket, and following packets in the socket do not need the request.

Figure 3: *Panorama* packet encapsulation.

Data plane forwarding via overlay source routing (SR). As shown in Figure 3, the *User Overlay SR Encap* component first derives raw IP packets from the upper applications, and then encapsulates each packet with a private *Panorama* header to encode the entire overlay route as a segment list of SE IDs. The component also maintains an SE-FWDR-MAP table preinstalled in the init process, and the table maps an SE ID to a set of IPs of the forwarders within the SE. Based on the table, a tunnel address is derived by hashing the inner packet's 5-tuple against the IP pool of the ingress SE, so that the packet can be sent through the underlay UDP tunnel hop by hop.

3.3 User/session grouping

In order to provide routing service for Internet-scale users, we need to group users into groups–a user group (UG) represents a set of users identified by an IP address space. As a common scale-reduction technique, user grouping is widely used in existing systems, e.g., based on geographic location in [13,26], autonomous system number (ASN) in [22], LDNS in [11,35], IP prefix in [23,28] or common router in traceroute in [27]. Ideally, the users in a UG should share the same best routing decision to a given destination. Therefore, user grouping in *Panorama* must account for the deployment of SEs.

In our approach, the finest grain unit is IP /24 prefix block, because /24 prefix is also the finest unit in BGP tables, and users in a /24 subnet also often share similar patterns of network performance. First for each /24 block, we rank all SEs in the ascending order of latency; then we combine all /24 blocks into UGs that have the same SEs in the top-M positions in the same order. We set $M = 2$ for UGs inside some countries where SEs are deployed more densely, and $M = 3$ for the other UGs. The user grouping can be adjusted based on latency measurement, but in practice, we find the grouping results are relatively stable. *Panorama* system covers around 750K active IP /24 blocks, and the grouping creates around 3K of UGs in the scenario with 30+ SEs.

Session group (SG). Based on UGs, SGs are simply given by $sg := (src\_ug, dst\_ug, service\_class)$. As a cross product, SG is on the order of millions. SG is the unit of granularity that *Panorama* routing works at, i.e., the routing and TE component makes routing decision for each SG. For SGs whose $src\_ug = dst\_ug$, the routing decisions are just direct Internet

path without using overlay, because the two ends are supposed to be close with each other. Next section describes how to compute routing for SGs that transmit data across different UGs.

3.4 Path adjustment

The overlay path of each session is determined when it is first created. But a session (e.g., a teleconference session) may last for long time, and the originally-allocated path may not be able to provide acceptable performance all through the session. Therefore, *Panorama* must support dynamic path adjustment capability during a session.

Specifically, to check if the current path is still feasible, the user's adjustment component uses a different interface to request the whole weighted path set, instead of just one path as when the session is created. Based on the path set, the component checks if the current path is within the set. If it is, the session will continue using the current path; and if no, the adjustment component will randomly select one based on the path weights to use. Path switching does not interrupt the session transmission, and it can be completed fast by simply encapsulating a different *Panorama* header.

Currently, the adjustment process is regularly triggered in every 10 minutes. We find the path is not switched in most cases, except for when the network has major degradation or the user is moving. We plan to design more sophisticated triggering algorithms based on network condition change and mobility in our future work.

3.5 Fast Reroute (FRR) from Failures

To guarantee the reliability, *Panorama* can safely reroute packets according to different failure locations on the route.

Case 1: Ingress Hop/SE failure. In this case, the *Panorama* user will fall back to use regular packet forwarding in the underlay network without activating *Panorama* function.

Case 2: En-Route SE failure. When the next-hop SE in packet header is failed, the packets will be directly sent to the egress SE via UDP tunnels.

Case 3: Egress failure. If the egress SE is failed or cannot be reached, the penultimate SE would serve as the egress. It locally removes the *Panorama* header and directly forwards the raw application packet to the final destination.

Case 4: Last hop failure. If the egress SE cannot reach the destination, it then reroutes the packet to the SE closest to the destination based on the destination IP. If the destination itself fails or becomes isolated, the *Panorama* user will be informed to deactivate *Panorama* function.

4 Overlay Routing and Traffic Engineering (TE)

This section gives the details of algorithms and implementation of the *Panorama* Overlay Routing and TE component, which computes the set of end-to-end paths and their weights to be used for each SG.

Specifically, the computation process consists of three phases. In Phase 1, based on the overlay network topology and real-time network measurement, the Top K shortest paths (KSP) in terms of latency connecting each pair of UGs are computed in a scalable, heuristic manner. In Phase 2, the paths that do not meet a given set of SLA constraints of each service class are filtered out from the KSP, to generate a feasible path set (FPS) for each SG. In phase 3, considering both traffic cost and link utilization, a simplified multi-commodity flow (MCF) problem is solved to compute the path weights in each FPS.

The three phases are executed sequentially. Since both Phase 1 and Phase 2 can be parallelized among SGs, the computation is split and assigned to a dozen of workers, and each worker is in charge of a subset of SGs. Phase 3 collects the results from distributed workers and conducts joint computation at a powerful super server. By default, the controller runs the above computation process every 5 minutes as a scheduling cycle. Based on computation results, the controller incrementally reconfigures and updates the group-decision tables in Route Servers. Operators can also trigger the computation and update manually.

4.1 Phase 1: Heuristic Top K shortest path (KSP)

As *Panorama* targets at providing premium real-time performance, latency is thus the most critical metric considered in *Panorama*. In this phase, the system first selects the Top K latency-wise shortest overlay paths of each SG based on real-time measurement.

A natural solution is to label the average latency as the weight of each overlay link, and run a classical (Top K) shortest path algorithm (e.g., Dijkstra, Yen [50]) to get the result. However, even though users have been abstracted into O(K) UGs, running classical KSP algorithms at such scale in real-time is computationally impossible.

One-big-switch abstraction of the backbone. To cope with the scalability issue in computing KSP, we propose a heuristic KSP algorithm tailored for *Panorama*. The algorithm relies on two facts observed in the overlay network: First, the backbone is the common part for every UG pair, so the computation inside the backbone can be reused. Second, SGs are highly independent with each other, so the computation for different SGs can be parallelized. The key idea of our heuristic KSP algorithm is to abstract the backbone as a big switch, and for each SG, the ingress/egress SE candidates are the ports of the

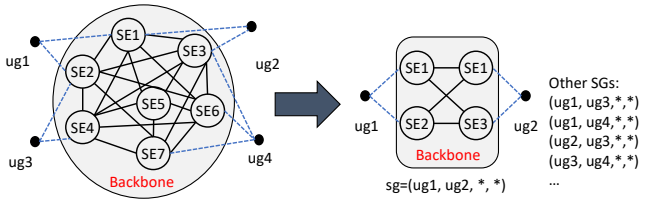


Figure 4: Example of the one-big-switch abstraction. The backbone is abstracted as a common big switch, where the KSP of any SE pair is computed in Step 1. The KSP of different UG pairs can be computed separately based on the backbone KSP.

K	3	5	7	9	10
AVG(# ingress)	2.16	2.87	3.36	3.72	3.85
AVG(# egress)	2.15	2.89	3.40	3.79	3.93

Table 1: Average numbers of ingress/egress SEs in the optimal KSP for diffferent K.

switch. Based on this abstraction, the algorithm consists of two following steps:

- Step 1 runs a classical loop-free KSP algorithm solely inside the backbone, and returns K' shortest backbone paths between any two SEs and the corresponding performance metrics.
- Step 2 searches the K shortest end-to-end paths for each SG among at most $N_i \times N_e \times K'$ conjunctional paths concatenating the src-ingress, ingress-egress and egress-dst segments, where N_i and N_e are the numbers of ingress and egress candidates of that SG respectively.

Figure 4 shows a concrete example of the one-big-switch abstraction idea in our heuristic KSP. The backbone can be abstracted as a common big switch, and different SGs can be computed separately upon it. For example $sg = (ug1, ug2, *)$, its ingress candidates are $\{SE1, SE2\}$ and egress candidates $\{SE1, SE3\}$. Assuming $K' = 2$ for the backbone KSP, then the algorithm just needs to search 7 end-to-end paths, 1 using only $SE1$ + 2 using $(SE1, SE3)$ + 2 using $(SE2, SE1)$ + 2 using $(SE2, SE3)$, and keeps the first K paths with lowest latency.

Theorem 1 *When $N_i = N_e = N_{total\_SE}$, and $K' \geq K$, the heuristic KSP can always achieve the optimal.*

Theorem 1 can be simply proved by route algebra, providing a sufficient condition to achieve the optimal in our algorithm. However, if all SEs are considered as candidates, the searching complexity as $O(N_i * N_e * K')$ will become computationally infeasible. Fortunately, we observe the following facts based on historical trace. First, as shown in Table 1, the average numbers of ingress/egress SEs in the optimal KSP ($K \leq 10$) is considerably small. Second, the optimal set is changed infrequently over time (i.e., retaining for tens of scheduling cycles) and gradually (i.e., changing only a small part at a time). Therefore, as long as the ingress/egress candidates are selected appropriately, the algorithm can achieve good optimality even with limited N_i and N_e.

Ingress/egress candidate selection based on reinforcement learning. We use the exploration-exploitation strategy to identify the ingress/egress candidates. The intuition of this strategy is to exploit currently-expected optimal decisions, while still giving chance to explore other potentially optimal decisions, thus adapting to dynamic change in the optimal set. The selections of ingress and egress candidates use the same algorithm and can be decoupled independently, so we only describe the ingress candidate selection process from now on. Exploration-exploitation strategy is originally proposed for the classic multi-armed bandit (MAB) problem. Compared with MAB, the difference in our scenario is that we need to select N_i "arms", instead of one each time. We use a modified version of discounted Upper Confidence Bound (UCB) [17] algorithm to determine the best candidates.

The discounted UCB is typically applied when the distributions of rewards remain constant over epochs and change at unknown time instants, so the weight of each observation keeps discounted along with time. Internet is well suited in such situation, as average performance can be affected by events like infrastructures outage or construction, new business relationships between ISPs and so on.

$$\begin{aligned} \mathrm{UCB}_{k,t} &= \hat{\mu}_{k,t} + \sqrt{\frac{\ln t}{n_{k,t}}}, \\ \hat{\mu}_{k,t} &= \frac{1}{n_{k,t}} \sum_{s=1}^{t} \gamma^{t-s} r_s(i) \mathbb{I}_{\{SE_k \in Candidate_s\}} \\ n_{k,t} &= \sum_{s=1}^{t} \gamma^{t-s} \mathbb{I}_{\{SE_k \in Candidate_s\}}. \end{aligned} \tag{1}$$

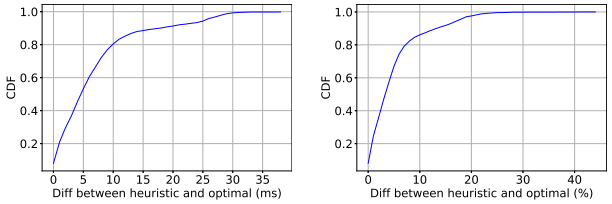
Equation (1) shows the discounted UCB for selecting SE_k as a candidate at time t. In the equation, $\hat{\mu}_{k,t}$ is the discounted empirical average, $\mathbb{I}_{\{SE_k \in Candidate_s\}}$ is a indicator function, with value 1 if SE_k is selected in the candidate at time s, and $\gamma \in (0,1)$ is time discount factor. $r_s(k)$ is the reward for choosing SE_k as a candidate at time s, which, in our scenario, is defined as the fraction of KSP paths that use SE_k as the ingress. For example, assuming at time s, the candidates are $\{SE_1, SE_2, SE_3\}$, and after calculation, the KSP are $\{SE_1, SE_1 - SE_2, SE_2 - SE_3 - SE_4\}$. Therefore, $r_s(1) = 2/3$, $r_s(2) = 1/3$, and $r_s(3) = 0$.

The selection of N_i candidates is split into two parts. Firstly, $N_i/2$ candidates are determined by selecting SEs with the highest $\hat{\mu}_{i,t}$, and then the other $N_i/2$ places are taken by the rest SEs with the highest $\mathrm{UCB}_{i,t}$. This is because we found if we choose the N_i candidates only based on $\mathrm{UCB}_{i,t}$, there exists cases that none of the candidates are in the oracle optimal set, leading to poor performance of the KSP. Our method that selects half by $\hat{\mu}_{i,t}$ and half by $\mathrm{UCB}_{i,t}$, increases the possibility of having the optimal SEs, while not giving up exploration, thus is much safer and better.

The computation complexity of this phase mainly lies in the searching process in Step 2. In practice, we set $N_i = N_e = 4$, $K' = 5$ and $K = 5$. Table 2 evaluates the computation time

# SG	100	1K	10K	100K	1M
Time (s)	0.18	0.21	0.40	2.96	23.70

Table 2: Completion time of Step 2 in our heuristic KSP for different numbers of SGs, in single-machine, single-core computation.



(a) (b)

Figure 5: CDF of latency increase (a) and ratio (b) between the heuristic KSP and the optimal KSP ($K = 5$).

of our heuristic KSP algorithms. Note that for clear illustration, the result is computed in a single machine with a single core, without parallelism optimization. We can tell that our algorithm is faster than classic algorithms by orders of magnitudes. Even without parallelism optimization, searching millions of SGs in Step 2 can be finished in 20 seconds. The fast execution not only stems from the significantly pruned complexity, but also is because the entire algorithm is executed by database operations, which have been greatly optimized by well-developed software. In practice, the computation is parallelized at multiple workers and cores, and each core is in charge of 10s of thousand SGs on average. The backbone KSP in Step 1 typically takes tens of milliseconds, so the entire of Phase 1 can be completed in one second.

While significantly reducing complexity, our algorithm does not lose optimality too much. Figure. 5 shows the evaluation of average latency of our heuristic KSP and the optimal KSP ($K = 5$) in 6.6 million of real session instances. On average, our heuristic algorithm increases latency by only 6.67ms and 4.49% than the optimal solution. Later we can see, following phases described in next subsections can further lower the impact of latency increase.

4.2 Phase 2: Feasible path set (FPS) by SLA filtering

The above end-to-end KSP is computed solely on the objective of latency, but sometimes a low-latency path may suffer from large packet loss or jitter that hurts service performance heavily. To avoid a path poor in any metrics, this phase excludes the paths violating the SLA requirements we assigned for each service class, and outputs an FPS for each SG.

An SLA requirement is defined as three upper bound thresholds of three performance metrics, namely, latency, jitter and loss. The SLA requirements are different across different *service_class* values, and independent to (src_ug, dst_ug). The values in SLA requirement are configured by operators in advance, and since *Panorama* deployed, they have been adjusted several times based on systematic measurement, analysis and feedback of application performance. Table 3 shows the SLA requirements used in the current version of *Panorama*. The diversity in service class empowers customers to optimize their service for performance and price. The Premium class users pay more for using exceptional better-performance paths with higher SLA requirement and a higher priority in bandwidth allocation (in Phase 3). The standard class users get control over cost, while still able to have SLA acceptable for RTC service. The Default class is mainly for free or trail users, and its values are also parts of the filtering algorithm.

service_class	priority	latency (ms)	jitter (ms)	loss
Premium	1	200	40	0.1
Standard	0	300	50	0.1
Default	0	400	50	0.1

Table 3: SLA requirement in current *Panorama*.

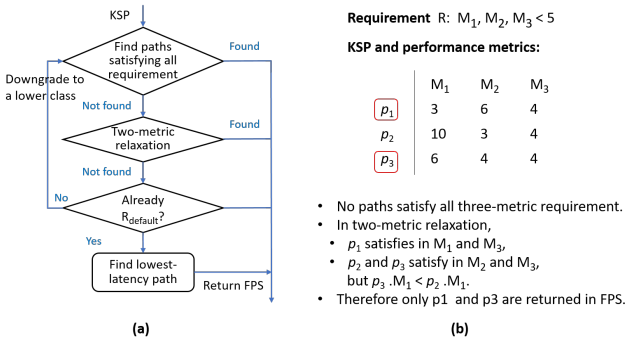


Figure 6: Multi-stage SLA filter.

The SLA filter works in multiple stages as shown in Figure. 6 (a). First, the SLA filter discards unsatisfiable paths and keeps the paths with all three metrics lower than the thresholds for each SG. In most cases, the filter process terminates at this stage and returns the FPS. But in some cases, due to either network degradation or too strict threshold (especially for long-distance SGs), there is no path in its KSP can satisfy the SLA. To ensure the FPS is not empty, the two-metric relaxation stage finds the paths that satisfy both the following conditions: 1) satisfying the SLA requirement in any two metrics, and 2) having the minimum value in the third unsatisfied metric. An example of the two-metric relaxation is shown in Figure.6 (b)). If still no path is found, the requirement is downgraded to a lower class all the way to $R_{default}$. Finally, if still no path found, the lowest-latency path is simply selected into the FPS, and an alarm will be triggered to notify operators to check if the network has any issues. Our measurements show that the last stage happens rarely: to only 0.47%. The multi-stage filter ensures at least one path in the FPS.

4.3 Phase 3: Weight allocation among the paths in FPS

After Phase 2, each SG obtains a set of paths that are expected to meet its SLA based on measurement results. However, the real network performance of a path may not be consistent with, and more seriously, may be worse than the estimated

performance, especially when the path faces major congestion. Therefore, instead of giving one path to each SG, the system computes the optimal weight allocation in FPS so that sessions in an SG can be distributed over those paths to minimize the congestion possibility.

In the literature, this problem is known as the multi-commodity flow (MCF) problem: given a capacitated network and a set of commodities (flow demands) between nodes, find an assignment of flows to paths that optimizes for some criterion, such as maximum overall throughput or max-min fairness [2]. In our scenario, the parameters of the MCF problem are as follows:

- **Network graph and capacities**: The nodes in the network graph G should include all SEs and UGs. The links include all backbone links (i.e., between SEs) and edge links (from a UG to an SE). The capacities of backbone links are estimated based on measurement, reflecting the maximum traffic that can be sent across that link. The edge links are difficult to be measured and they share the same port to an SE. To model the capacities of edge links, we extend the graph to G' by adding a virtualized node VN for each SE, as shown in Figure 7. The link capacity from each UG to VE is infinity, and that from VN to SE is the network card rate.

- **Commodities**: The commodities are all SGs, and the demand d_i of sg_i is estimated by measuring the average traffic rate of sg_i over the last scheduling cycle.

- **Constraint paths**: Constraining usable paths in a given set can greatly simplify the computation of MCF. The constraint paths of an SG is a modified version of FPS aligning with the extended topology G'.

- **Objectives**: Since the main goal here is to balance load and minimize congestion, the optimization objective is cost-weighted maximum link utilization (C-MLU). Here the cost reflects the bandwidth price variation of different links.

We solve the MCF problem using linear programing (LP), as formulated in equation (2), with terms shown in Table 4. Specifically, the allocation runs the LP twice, separately for SGs in priority order. After high-priority SGs are allocated, the allocation is removed from the remaining link capacities.

$$
\begin{aligned}
\text{minimize} \quad & \text{C}-\text{MLU} \\
\text{s.t.} \quad & U_e = \frac{\sum_{sg_i \in SG} \sum_{p_j \in \text{FPS}_\text{i}} w_{ij} \cdot d_i}{cap_e}, \\
& U_e \cdot cost_e \leq \text{C}-\text{MLU}, \quad \forall \text{e} \in \text{E}, \\
& \sum_{p_j \in \text{FPS}_\text{i}} w_{ij} = 1, \quad \forall i | sg_i \in SG, \\
& w_{ij} \geq 0, \quad \forall i, j | sg_i \in SG, p_j \in \text{FPS}_\text{i}.
\end{aligned}
\tag{2}
$$

Optimization. The LP in our allocation is much less complex than the LP used to solve traditional MCF, but solving

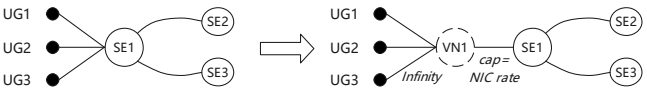


Figure 7: Extension of the topology graph to G' with virtual nodes to model edge link capacities. The link capacity from each UG to VN_1 is infinity, and that from VN_1 to SE1 is the network card rate of SE1.

	Variable	Definition
Input	$G'(V,E)$	Extended topology graph
	$\{cap_e\}$	Capacity of links $\{e\}$
	$\{cost_e\}$	Cost of links $\{e\}$
	$\{\text{FPS}_\text{i}\}$	FPS of $\{sg_i\}$
	$\{d_i\}$	Predicted demands of $\{sg_i\}$
Auxiliary	p_j	Path j
	U_e	Utilization of link e
	C-MLU	Maximum link utilization
Output	$\{w_{ij}\}$	{weight of path p_j in FPS_i}

Table 4: Terminologies in equation (2).

O(million) SGs still can takes up to several minutes. To this end, we optimize the scale by running the LP only for the "active" SGs, which are defined as the SGs with demand over a certain threshold. The number of "active" SGs is typically at the scale of O(10K), which significantly speeds up the LP process. As to the allocation for "inactive" SGs, the weights are simply divided equally among its FPS. In addition, before running the LP, we trim down the FPS to include at most 3 paths with minimum hops. This can also save traffic cost, as more hops mean more traffic flowing in and out the overlay infrastructure, resulting in more money needs to be paid for bandwidth resource. We use a customized optimizer software to solve the LP, and we set a time constraint (e.g., 1 minutes). When timeout, the optimizer can returns the currently best solution. In this way, even in the face of busy hours with substantial "active" SGs, the allocation phase can still be finished in definite time.

5 Evaluations

This section demonstrates the benefits of *Panorama* in real-world deployment environment.

5.1 Real-world deployment of *Panorama*

Panorama has been implemented and deployed in one of the largest real-time overlay networks in the Internet. The overlay network has been built since three years ago, and now it already covers SEs deployed in 30+ locations globally, and carries a colossal amount of real-time interacting applications traffic for multiple content providers (CPs). Each of our SE can access the public Internet. To provide redundancy and inter-ISP optimization, half of SEs are BGP multihoming, i.e.,

by peering with more than one ISPs. In each SE, multiple forwarding instances (a.k.a. forwarders) are deployed for the purpose of load balancing and fault tolerance. Each forwarder is allocated with an elastic IP (EIP). SEs can process and relay traffic at greatly high rate: a single forwarder can process 1 million packets per second, forward traffic at rate of 10 Gbps. On top of the overlay network, the *Nearby* paradigm was first adopted, until its inefficiency was recognized based on the operation and maintenance observation. Since Dec. 2021, *Panorama* has been tested and gradually used to replace the previous *Nearby*-based system.

We develop a hands-on SDK for application developers to use *Panorama* and the overlay networking service in a simple way. The SDK supports a variety of applications with diverse requirements, and can be applied at different kinds of endpoint devices including user terminals (e.g., smart phone), PCs and backend servers (e.g., CDN, SFU and Http server). As for compatibility, the SDK has versions supporting most mainstream platforms, including Android, iOS, Linux, and Web.

Evaluation methodology. To evaluate the performance of *Panorama*, we collect 81 million session instances in deployment environment, where the session instances are distributed among users in 66 countries. In order to fairly compare routing paradigms among the three: *Panorama*, *Nearby* and direct Internet, we keep sessions only if this session's belonging SG has performance data for all three routing paradigms within a 5-minute window. We accept performance data of a session either directly collected from the SDK RUM, or constructed using network tomography [10] based on real-time measurement within the belonging SG. After the selection, 6.6 million sessions in total are eligible for participating in the evaluation, and they belong to 6K "active" SGs. We also synthesize an Oracle-latency-optimal path for each SG purely based on measurement results.

5.2 *Panorama* improves network performance

Improvement on general network metrics. Figure 8 shows the network performance CDF of three metrics respectively, for different routing paradigms. From Figure 8 (a), we can tell that the 50^{th}, 90^{th}, and 99^{th} of *Panorama* is 10.2%, 17.5%, and 23.0% faster than *Nearby*, respectively, and 18.2%, 27.9%, and 32.9% faster than direct Internet, respectively. *Panorama* outperforms the previous *Nearby* approach by jointly optimizing both user access and backbone, while considering link utilization to avoid congestion. Figure 8 (b) and (c) shows *Panorama* can achieve better loss and jitter performance than *Nearby* and even the oracle. This is because the oracle method is purely based on latency, but our approach accounts for multiple objectives more than just latency, thus can find paths good in all three metrics.

One may notice that *Panorama* sometimes can bring negative gains compared with direct Internet. Indeed, for some short-distance sessions, there is no need to use the overlay network which incurs extra latency and jitter. However, most of the negative gains would not be perceived by users, as they happen only in <100ms sessions. Therefore, the *negative gain will not affect the user experience*. Currently, only the SGs within the same UG are using direct routing, and in the future we plan to design more advanced mechanism to allow more closely-located UGs to use direct Internet.

Figure 10 shows the latency "sub optimality" of both *Panorama* and *Nearby*, which is defined as the latency increase ratio compared with the oracle optimal [22]. The medians of "sub optimality" of *Panorama* and *Nearby* are 3.83% and 16.51%, respectively. As we can see, the algorithm of *Panorama* shows great efficiency by increasing optimality by 76.80%, and achieving close performance indicated by the oracle optimal.

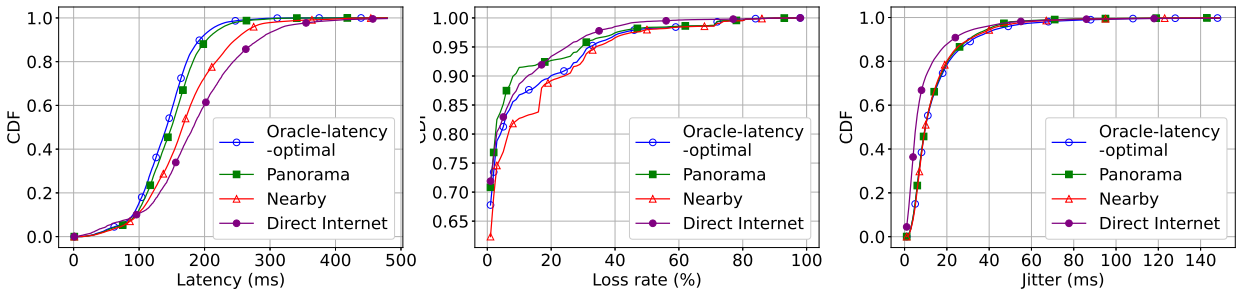
Improvement on SLA satisfaction. The main benefit of *Panorama* is not to turn a good-quality session into a great-quality one, but to change more bad-quality sessions into good-quality ones, as only the latter can truly improve the user experience to meet the SLA requirement. Figure 9 shows the portions of bad-quality sessions under different performance metric conditions. We can tell that *Panorama* reduces >300ms sessions by 97.8% than *Nearby*, and 99.3% than direct. More importantly, in *Panorama*, the experienced latency of 96.34% sessions can be limited under 200ms, and that of 99.98% of the sessions under 300ms. Bad-quality sessions with large loss and jitter can also be reduced to improve application performance.

Table 5 illustrates the SLA satisfaction for different approaches. We can tell *Panorama* increases satisfaction by up to 27.0%, 2.5%and 4.3% than *Nearby* approach, and 88.02%, 5.5% and 1.6% than direct Internet for Premium, Standard and Default service classes respectively. Among all the classes, *Panorama* improves the Premium service the most, therefore it plays a critical role to compete with other commercial real-time overlay networking. According to Table 3, the SLA requirement for Premium service is the most strict, especially in terms of latency: 200ms already exceeds the physical limitation of some SGs. By carefully picking the best-performance paths and allocating appropriate bandwidth, *Panorama* can still achieve good satisfaction, showing the ability to support more time-sensitive applications such as online gaming.

$SLA\_satisification(\%)$	*Panorama*	Nearby	Direct
Premium	76.60	60.33	40.74
Standard	83.96	79.94	79.56
Default	84.05	80.59	82.69

Table 5: SLA satisfaction for different routing approaches.

Improvement by locations. A session's network performance can depend on the locations of its communication pairs. As *Panorama* has been deployed and used around the



(a) (b) (c)

Figure 8: Network performance CDF of three metrics respectively: (a) Latency, (b) Loss, and (c) Jitter.

Latency	Oracle	Panorama	Nearby	Direct
>200ms	3.55%	3.66%	23.12%	27.74%
>300ms	0.02%	0.02%	0.92%	3.03%
>400ms	0.00%	0.00%	0.03%	0.51%

(a) Latency

Loss	Oracle	Panorama	Nearby	Direct
>20%	7.98%	5.15%	9.15%	8.72%
>50%	0.47%	0.34%	0.62%	1.79%
>70%	0.22%	0.16%	0.25%	0.55%

(b) Loss

Jitter	Oracle	Panorama	Nearby	Direct
>50ms	1.41%	0.86%	1.34%	4.13%
>100ms	0.33%	0.24%	0.30%	0.91%
>250ms	0.07%	0.04%	0.03%	0.26%

(c) Jitter

Figure 9: Portions of bad-quality sessions under different performance metric conditions: (a) Latency, (b) Loss, (c) Jitter.

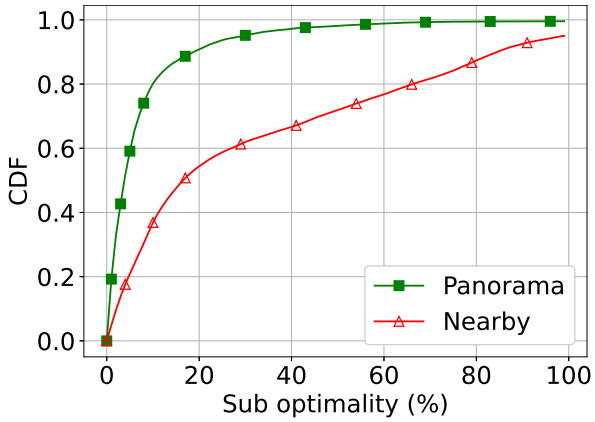


Figure 10: Latency "sub optimality" of *Panorama* and *Nearby*.

whole world, there is abundant data to further dissect the improvement of *Panorama* by locations. We group users into five representative regions: EA with 2 countries represents East Asian, AS with 21 countries represents the rest of Asian and a country in Oceania, EU with 25 countries is Europe, AM with 9 countries include both North and South America, and AF with 9 countries is Africa. The session performance is then aggregated and calculated by the regions.

Figure 11 shows the average latency comparison across different regions. To better illustrate, we aggregate the latency results for both directions of each region pair, e.g., AM-AF represents session results either from America to Africa or from Africa to America. Figure 12 shows the average latency comparison within each region. From the figures we can tell there is a substantial diversity on improvement across different regions. The improvement of *Panorama* over *Nearby* can achieve 44.61% for inter-region sessions from EU to AM, and 40.97% for intra-region sessions in EA. Besides decreasing the average latency, *Panorama* can also provide more stable performance by narrowing down the confidence interval by up to 35.32% for AS-AM inter-region sessions, and 49.97% for EA intra-region sessions. Both figures demonstrates that *Panorama* can achieve the almost equivalent average latency to the oracle wherever the session is sent from or to, and *Panorama* can successfully control the average latency below 180 ms for any inter-region sessions, and below 150 ms for any intra-region sessions.

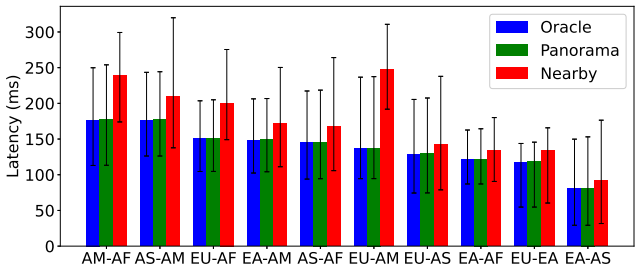


Figure 11: Latency comparison for inter-region sessions.

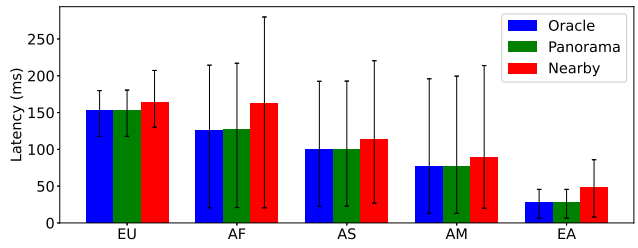


Figure 12: Latency comparison for intra-region sessions.

Comparison with another real-time overlay networks. To better demonstrate *Panorama*'s practical benefit, we compare *Panorama* with another widely-used commercial real-time overlay networking service–Agora SD-RTN [1] in a small-scale testbed. The testbed include 21 users spread over 18 countries, and each user is running on a Linux virtual machine with both *Panorama* and Agora SDKs installed. We choose Agora as a comparison because Agora SD-RTN has competitive performance in providing high QoE for real-time communications in the world, and it has a similar deployment scale as *Panorama*, so that we can minimize the infrastructure affect and focus on comparison of the routing system. We develop a script on each user to continuously send UDP packets to each other and echo back, using Agora and *Panorama* services in turns, and we can record the latency of each UDP packet and the associated overlay service.

Figure 13(a) shows the latency improvement (decrease) of *Panorama* over Agora for the same transmission. We can tell that both systems have their superiority in different cases respectively, but in most cases (80%), *Panorama* outperforms Agora, by up to 250 ms. Figure 13(b) further shows the latency comparison of the three methods. We can observe *Panorama* achieves 30.6% improvement (i.e., reduction) on median latency compared to Agora. More interestingly, the 95^{th} latency values (in ms) of Agora, *Panorama* and *Nearby* are 268, 191 and 280 respectively, indicating *Nearby* is worse than Agora, while *Panorama* surpasses Agora. The result demonstrates the effect of *Panorama* from an algorithmic perspective rather than an infrastructure-based one.

5.3 *Panorama* reduces bandwidth cost and improves load balancing

The bandwidth cost of operating the overlay network is confidential information, so we use the overlay hop numbers to reflect the cost. Figure 13(c) shows the session distribution in different hop numbers along the paths. In *Panorama*, about 65.1% sessions use only one hop, and about 93.1% use no more than two hops. In *Nearby* approach, each session requires at least two hops for ingress and egress (unless they are the same). The average SE numbers of *Panorama* and *Nearby* are 1.43 and 2.57. Roughly, *Panorama* can save bandwidth cost by 44.3%.

Figure 14 shows the normalized load distributions in all SEs, ordered by the traffic load. We use standard deviation (STDDEV) to reflect the balance level, and the STDDEVs of *Panorama* and *Nearby* are 0.10 and 0.22 respectively. As a result, *Panorama* balances the load 54.5% more than *Nearby*, achieving much lower and more balanced load than *Nearby* approach. To clarify, SE load depends on the geographical distribution of users and sessions (e.g., some hot spots), so it's impossible to accomplish equally balanced.

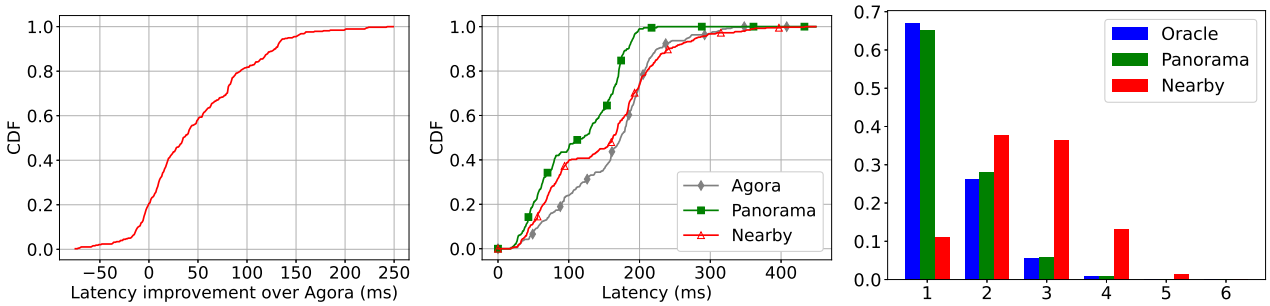
6 Discussion

This section summarizes some experience we learned in implementing and deploying *Panorama*.

Public Internet overlay is an important addition to private networks. Since building large-scale private networks can take exponential cost for private circuits if strong network connectivity is desired, we adopt another deployment strategy using hybrid connectivity, where private links are used on critical overlay links in a "circle + star"-like topology, and Internet links connect every two SEs in a full mesh manner. Internet link and private link can coexist between two SEs, implemented by installing two NICs in a forwarder. Public Internet overlay is a good addition to private links, with proved benefits including offloading traffic from private links, enhancing network capacity with low cost, and providing shortcut paths even better than private links. The hybrid networking strategy has been used in industrial overlay networks like SD-RTN by Agora [1], and SD-WAN service providers like Aryaka [36]. We believe the "overlay + hybrid connectivity" is not only a preliminary strategy for low cost networking, but also a trend for building large-scale WANs in the future.

There is inconsistency between active measurement and real experience. As a main input of routing and TE, the measurement DB basically comes from active measurement at SEs. However, there can be huge inconsistency between measurement and real experience, e.g., the measured latency of a link is small, but when sending real traffic, the latency becomes large, leading to suboptimal in routing decisions. The purpose of Phase 3 in our computation is to alleviate the inconsistency by minimizing congestion, but it can not be fully eliminated. We observe the experience of UDP packets, especially low-rate UDP packets, is mostly consistent with measurement, but http packets may have huge difference. In the future work, we plan to incorporate with more passive measurement data as input, and enrich active measurement to approximate real experience, e.g., by using the same protocol, same encapsulation and so on.

Machine learning (ML) techniques must be used in a safe way. Recently, substantial efforts have been devoted to use ML techniques for traffic control. Despite the appealing benefit, their practical deployment faces difficulty. For example, Via [22] applies an online exploration-exploitation strategy to select desired relay paths. Therefore, there inevitably will be some "unluck" sessions in the exploration phase using undesirable paths with extremely poor performance. This is against the objective of Panorama to improve the SLA satisfaction of all users. The fundamental reason is the unpredictable behaviors in ML methods. *Panorama* only uses ML in intermediate processes (i.e., to provide ingress/egress candidates), thus prevents such risk in final decisions. We will introduce more ML techniques into *Panorama* (e.g., for user grouping and measurement) in the future.



(a) (b) (c)

Figure 13: (a) CDF of the latency improvement (decrease) of *Panorama* over Agora. (b) CDF of latency for *Panorama*, *Nearby* and Agora. (c) Overlay hop-number distribution

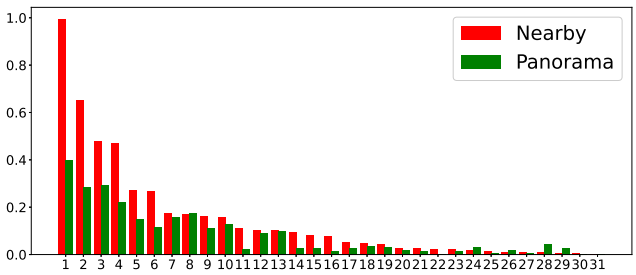


Figure 14: Normalized load distributions among SEs.

7 Related Work

In Section 2.3, we have extensively discussed the closely related overlay routing solutions. Here we overview some other related ideas which have not been discussed elsewhere.

Traffic engineering (TE). WAN TE has been an important topic for the past decades, and tons of solutions have been proposed in a variety of contexts. In recent years, to improve the transmission efficiency of bulk-data transfers across datacenters(DCs), a number of solutions, such as SWAN [20], B4 [21] and SMORE [24], are proposed. Most of the solutions under this category fall short of scalability (i.e., operating on hundreds of nodes rather than millions), and their operation over aggregated traffic at coarse timescale does not meet the stringent latency requirement for each individual task. Footprint [28] and Entact [51] are two TE schemes for delivering online services over integrated infrastructure which consists of multiple networks. They consider the traffic of mixed online services and mainly target the user-DC scenario, rather than the scenario to end-to-end deliver real-time interactive content.

Content delivery network (CDN) system. Another typical routing system that support Internet-scale users is CDN. For example, Akamai's end-user mapping system [11] and Microsoft's Odin system [8] use the DNS protocol to route each client's request to a "proximal" server that serves the requested content. Unfortunately, the techniques to deploy edge servers close to users for caching static content does not fit our problem of end-to-end delivering real-time content.

Machine Learning in TE. Following the success of Deep Learning in many fields of computing, it is inevitable that a significant amount of effort is devoted to researching its application in the field of network TE in the past few years [48]. MARL-GNN [6] proposes an ML-based TE framework that combines RL and GNN to generate good OSPF link weight settings to minimize MLU with shortest path routing. Many of the research efforts are dedicated to leveraging DRL, a DL technique that utilizes historical information for future decision-making. DRL has been applied to many different TE scenarios, including intradomain TE [7, 45, 49], Software Defined Wide Area Networks (SD-WAN) [29, 32], SDN-based Optical Transport Network [3], and Data Center TE [12]. However, these line of work are seldomly deployed in practice, as the performance of the ML-based method heavily hinges on arbitrarily determined reward values, which may output unpredictably bad decisions as we discussed in Section 6.

8 Conclusion

Network performance is critical to Internet real-time applications nowadays, yet still far from satisfiable under the traditional Internet framework. Given its importance, substantial efforts in both infrastructure and algorithms have been devoted to improving the network performance. Despite considerable progress, existing approaches still suffer from major limitations in taking full use of the infrastructure, mainly because of the scalability issue in handling millions of global users. This paper presents *Panorama*, which pushes the SD control domain to every end user, and optimizes routing in user-backbone integration. *Panorama* systematically addresses the scalability challenge in multiple dimensions with coherent combination, and has been proved to perform well in real-world environment. While our approach does not generalize to all WANs or overlay networks, we hope that our experience summarized from our practice with real systems can inform future work in these domains.

References

[1] What makes the agora sd-rtn the most reliable, scalable and efficient global engagement platform in the world. Technical report.

[2] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.

[3] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *arXiv preprint arXiv:1910.07421*, 2019.

[4] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 131–145, 2001.

[5] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. SIGCOMM '02.

[6] Guillermo Bernárdez, José Suárez-Varela, Albert López, Bo Wu, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Is machine learning ready for traffic engineering optimization? In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2021.

[7] Guillermo Bernárdez, José Suárez-Varela, Albert López, Bo Wu, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Is machine learning ready for traffic engineering optimization? In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2021.

[8] Matt Calder, Ryan Gao, Manuel Schröder, Ryan Stewart, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. Odin: Microsoft's scalable fault-tolerant {CDN} measurement system. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 501–517, 2018.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 2002.

[10] Rui Castro, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. Network tomography: Recent developments. *Statistical science*, pages 499–517, 2004.

[11] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. *ACM SIGCOMM Computer Communication Review*, 45(4):167–181, 2015.

[12] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 191–205, 2018.

[13] Gloria Ciavarrini, Valerio Luconi, and Alessio Vecchio. Smartphone-based geolocation of internet hosts. *Computer Networks*, 116:22–32, 2017.

[14] Cloud VR Network Solution White Paper. https://www.huawei.com/minisite/pdf/ilab/cloud_vr_network_solution_white_paper_en.pdf.

[15] AWS Global Accelerator. https://aws.amazon.com/global-accelerator.

[16] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. C3: Internet-scale control plane for video quality optimization. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 131–144, 2015.

[17] Aurelien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*, 2008.

[18] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. In *NSDI*, volume 4, pages 9–9, 2004.

[19] Osama Haq, Mamoon Raja, and Fahad R Dogar. Measuring and improving the reliability of wide-area cloud paths. In *Proceedings of the 26th International Conference on World Wide Web*, pages 253–262, 2017.

[20] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pages 15–26, 2013.

[21] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[22] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, et al. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 286–299, 2016.

[23] Yuchen Jin, Sundararajan Renganathan, Ganesh Ananthanarayanan, Junchen Jiang, Venkata N Padmanabhan, Manuel Schroder, Matt Calder, and Arvind Krishnamurthy. Zooming in on wide-area latencies to a global cloud provider. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 104–116. 2019.

[24] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 157–170, 2018.

[25] Jinu Kurian and Kamil Sarac. A survey on the design, applications, and enhancements of application-layer overlay networks. *ACM Comput. Surv.*, 2010.

[26] Youndo Lee and Neil Spring. Identifying and aggregating homogeneous ipv4/24 blocks with hobbit. In *Proceedings of the 2016 Internet Measurement Conference*, pages 151–165, 2016.

[27] F Thomson Leighton, Ravi Sundaram, Matthew Levine, and Adrian Soviani. Method for generating a network map, July 31 2007. US Patent 7,251,688.

[28] Hongqiang Harry Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitendra Padhye, and Ming Zhang. Efficiently delivering online services over integrated infrastructure. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 77–90, 2016.

[29] Libin Liu, Li Chen, Hong Xu, and Hua Shao. Automated traffic engineering in sdwan: Beyond reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 430–435. IEEE, 2020.

[30] Cristian Lumezanu, Randolph Baden, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Symbiotic relationships in internet routing overlays. In *NSDI*, pages 467–480, 2009.

[31] The metaverse is the next evolution of social connection. https://about.facebook.com/meta/.

[32] Bashir Mohammed, Mariam Kiran, and Nandini Krishnaswamy. Deeproute on chameleon: Experimenting with large-scale reinforcement learning and sdn on chameleon testbed. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–2. IEEE, 2019.

[33] Matthew K. Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. Practical, real-time centralized control for cdn-based live video delivery. SIGCOMM '15.

[34] Akihiro Nakao, Larry Peterson, and Andy Bavier. A routing underlay for overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 11–18, 2003.

[35] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[36] Aryaka. https://www.aryaka.com.

[37] Google Cloud Network Service Tiers. https://cloud.google.com/network-tiers.

[38] Video Conferencing Statistics 2020. https://www.uctoday.com/collaboration/video-conferencing/video-conferencing-statistics/.

[39] Stefan Savage, Thomas Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, et al. Detour: Informed internet routing and transport. *Ieee Micro*, 19(1):50–59, 1999.

[40] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The end-to-end effects of internet path selection. SIGCOMM '99.

[41] Ramesh K Sitaraman, Mangesh Kasbekar, Woody Lichtenstein, and Manish Jain. Overlay networks: An akamai perspective. *Advanced Content Delivery, Streaming, and Cloud Services*, 51(4):305–328, 2014.

[42] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H Katz. Overqos: An overlay based architecture for enhancing internet qos. In *NSDI*, volume 4, pages 71–84, 2004.

[43] Yasuko Sugito, Shinya Iwasaki, Kazuhiro Chida, Kazuhisa Iguchi, Kikufumi Kanda, Xuying Lei, Hidenobu Miyoshi, and Kimihiko Kazui. Video bit-rate requirements for 8k 120-hz hevc/h.265 temporal scalable coding: experimental study based on 8k subjective evaluations. *APSIPA Transactions on Signal and Information Processing*, 2020.

[44] Shengwen Tian, Jianxin Liao, Tonghong Li, Jingyu Wang, and Guanghai Cui. Resilient routing overlay network construction with super-relay nodes. *KSII Transactions on Internet & Information Systems*, 11(4), 2017.

[45] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to route. In *Proceedings of the 16th ACM workshop on hot topics in networks*, pages 185–191, 2017.

[46] Video Conferencing Network Requirements. https://www.videonations.co.uk/resources/video-conferencing-news/video-conferencing-network-requirements/.

[47] 74 Virtual Reality Statistics You Must Know in 2021/2022: Adoption, Usage Market Share. https://financesonline.com/virtual-reality-statistics/.

[48] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. Machine learning for networking: Workflow, advances and opportunities. *Ieee Network*, 32(2):92–99, 2017.

[49] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1871–1879. IEEE, 2018.

[50] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

[51] Zheng Zhang, Ming Zhang, Albert G Greenberg, Y Charlie Hu, Ratul Mahajan, and Blaine Christian. Optimizing cost and performance in online service provider networks. In *NSDI*, pages 33–48, 2010.